

RAPPORT DE STAGE



Étudiante :	SHCHERBAKOVA Kateryna
Cours :	Master 1 mention Informatique
Parcours :	Ingénierie logicielle
Établissement :	ISTIC (UFR INFORMATIQUE-ÉLECTRONIQUE)
Enseignant Réfèrent :	BELLEANNÉE Catherine



Entreprise :	Alkante
Service :	PÔLE GÉOMATIQUE
Adresse :	Parc d'activités des Vents d'Ouest, 4 Rue Alain Colas, 35530 Noyal-sur-Vilaine
Tuteur professionnel :	SUZANNE Anthony

Rennes, 2023

CONTENU

DESCRIPTION DU STAGE.....	1
1. Présentation de l'entreprise.....	1
1.1. Description de l'entreprise.....	1
1.2. Description du service.....	1
1.3. Organisation du travail.....	1
2. Sujet du stage.....	2
3. Travail réalisé durant le stage.....	3
3.1. Installation et mise en place du projet.....	3
3.2. Écrire des tests d'API.....	3
3.3. L'idée de créer un générateur de tests.....	4
3.4. Structure du générateur de test.....	5
3.5. Options de commande.....	7
3.6. Création d'un bundle.....	7
3.7. Mise en œuvre du projet.....	8
3.8. Résultats du générateur de test.....	8
4. Poursuite du développement du projet de stage.....	9
5. Bilan du stage.....	9
RÉSUMÉ.....	11
ANNEXES.....	12
Annexe 1. Schéma d'entrée et de sortie de la classe Json.....	12
Annexe 2. Rapport de bogue.....	15

DESCRIPTION DU STAGE

1. Présentation de l'entreprise

1.1. Description de l'entreprise

Mon stage s'est déroulé dans une petite entreprise prometteuse, Alkante. L'entreprise travaille dans le domaine de la technologie numérique et est spécialisée dans trois domaines :

1. La géomatique sur mesure, les projets web et mobiles ;
2. Intégration de solutions spécialisées telles que l'internet des objets, les systèmes d'information géographique, la gestion foncière, etc ;
3. Hébergement et gestion de plateformes web.

« Au cœur de nos engagements pour apporter des réponses opérationnelles aux enjeux de nos clients figurent l'efficacité technologique et l'indépendance numérique pour une transition digitale éco-responsable. »

Alkante. « Qui sommes-nous ? ». [En ligne] Disponible sur: <https://www.alkante.com/accueil/qui-sommes-nous> [Consulté le 28 juin 2023].

1.2. Description du service

Le département de l'entreprise où j'ai effectué mon stage s'appelle "Pôle géomatique". Les équipes de ce service développent des projets autour de trois axes principaux : portails géographiques, applications métiers et Internet des objets.

1.3. Organisation du travail

Alkante se compose d'équipes autonomes réparties en départements thématiques et en groupes de projet. Pour garantir la cohérence, des réunions régulières sont organisées pour discuter des objectifs, des plans, des problèmes et des solutions communes.

La coopération est basée sur une atmosphère de travail amicale. L'entreprise s'adapte au rythme de chaque employé et offre une grande flexibilité en termes de télétravail.

2. Sujet du stage

Le titre de mon stage était "Mise en œuvre de tests outillés pour une plateforme cartographique". Il s'agissait d'écrire des tests unitaires, des tests d'intégration, des tests fonctionnels et autres pour les différents modules de la plateforme appelée "Prodige". Les tests devaient être implémentés en PHP en utilisant le framework PHPUnit. Le développement devait se faire dans un projet dont le backend était écrit avec le framework Symfony.

La plateforme Prodige se compose de nombreux modules. Ma tâche consistait à écrire des tests pour le module appelé "Admin", qui est responsable de l'administration des autres modules de la plateforme. Pour mettre en œuvre les fonctions de cartographie nécessaires, la plateforme utilise une API pour créer, modifier, supprimer des données, etc. Les tests de la plateforme étaient basés sur l'API : il était nécessaire de vérifier différentes requêtes et leur résultat.

Au cours des deux premières semaines, j'ai travaillé à la mise en œuvre des objectifs. Cependant, au cours de la troisième semaine, j'ai proposé de modifier le but et les objectifs finaux du stage. Pendant le temps restant, j'ai écrit un générateur de test. Sa fonctionnalité permet au développeur d'entrer une commande dans la console qui contient le nom de la classe souhaitée. Ensuite, une classe est créée, qui contient le code de tous les tests requis pour les méthodes de la classe. Le raisonnement détaillé pour changer l'objectif du stage, ainsi que le principe du générateur de tests sont décrits dans "Travail réalisé durant le stage".

J'ai terminé le générateur de tests trois semaines avant la fin du stage, de nouveaux éléments ont donc été ajoutés à l'objectif final. Il fallait créer un bundle qui permettrait d'incorporer les fonctionnalités du générateur de test dans divers projets de l'entreprise à l'aide d'une seule commande de la console. Un fichier de configuration devait également être créé pour modifier les paramètres du générateur de test.

3. Travail réalisé durant le stage

3.1. Installation et mise en place du projet

Avant d'écrire les tests, le projet de plateforme Prodige a dû être installé et configuré :

1. Clonez le projet en utilisant la commande git "git clone"

```
git clone git@gitlab.alkante.com:alkante/ppr_prodige_admin.git
```

2. Définir la structure de dossier correcte pour le projet
3. Mettre en place les dépendances nécessaires
4. Construire le projet à l'aide de la commande Docker

```
docker-compose-dev up -d
```

Après avoir construit le projet, il faut accéder au conteneur Docker à l'aide d'une commande de la console pour exécuter les tests :

```
docker exec -it -u www-data ppr_prodige_admin_web bash
```

3.2. Écrire des tests d'API

L'un des axes du Pôle géomatique où j'ai effectué mon stage était de développer une API pour l'interaction avec les ressources de la plate-forme. Ma tâche consistait à tester cette API pour les classes de la plateforme Prodige.

Le test des requêtes API a été effectué sur la base du schéma d'entrée et de sortie Json de la classe testée. Les paramètres tels que les champs obligatoires, le type de champ, le nom du champ, etc. ont été extraits du schéma d'entrée pour les tests. Ces paramètres ont été utilisés comme entrées pour envoyer la requête. D'autre part, le résultat obtenu à partir de la requête a été comparé au schéma de sortie Json. Vous pouvez voir un exemple du schéma d'entrée et de sortie de la classe Json dans annexe numéro 1.

Au cours des deux premières semaines, j'ai rédigé 11 types de tests pour trois classes. Ces tests sont les suivants :

1. testCreateItem : test d'une requête POST, création d'une ressource avec seulement les champs obligatoires ;
2. testCreateItemWithAllFields : test d'une requête POST, création d'une ressource avec tous les champs possibles ;
3. testCreateItemWithNonExistingFields : test d'une requête POST, création d'une ressource. Dans ce cas, si l'objet contient un champ faisant référence à un autre objet, cette référence est mal spécifiée ;
4. testGetCollection : test d'une requête GET, obtention de toutes les ressources ;
5. testGetItem : test d'une requête GET, obtention d'une ressource.
6. testGetNonExistingItem : teste d'une requête GET, récupération d'une ressource qui n'existe pas ;
7. testUpdateItem : test d'une requête PATCH, modification des champs d'une ressource ;
8. testUpdateNonExistingItem : test d'une requête PATCH, modification d'une ressource inexistante ;
9. testUpdateItemWithWrongHeaders : test d'une requête PATCH, modification d'une ressource en spécifiant des en-têtes non valides ;
10. testDeleteItem : test d'une requête DELETE, suppression d'une ressource ;
11. testDeleteNonExistingItem : test d'une requête DELETE, suppression d'une ressource inexistante.

Pour exécuter les tests, il faut se connecter au conteneur docker et utiliser la commande PHPUnit :

```
php bin/phpunit pathToTestClass;
```

3.3. L'idée de créer un générateur de tests

En écrivant des tests pour trois classes, on a remarqué que la structure des mêmes tests est très similaire. Les différences entre les tests des différentes classes sont les noms des champs, les types de données, la présence et le nombre de champs obligatoires, etc.

Ci-dessous se trouve une partie du code pour tester la création d'une ressource pour deux classes : Domain et Layer. La couleur indique les différences entre ces tests :

```

/**
 * create one Domain (201)
 * @group testPost
 */
public function testCreateDomain(): void
{
    //-----
    // initialize required fields with the random values depending on their types
    //-----
    $DomainName = $this->generateRandomString();

    //-----
    // initialize required fields which are references to other entities
    //-----
    $response = $this->createClientWithCredentials('/api/rubrics');
    $this->assertResponseIsSuccessful();
    $requiredRefItem = '/api/rubrics' . '/' .
        strval($response->toArray()['hydra:member'][0]['id']);
    $DomainRubric = $requiredRefItem;

    //-----
    // create Domain with only required fields
    //-----
    $headers = ['content-type' => 'application/ld+json'];
    $body = [
        'name' => $DomainName,
        'rubric' => $DomainRubric,
    ];

    $response = $this->createClientWithCredentials($this->url, 'POST', $headers,
    $body);
    $this->assertResponseStatusCodeSame(201);

    self::$ItemId = $response->toArray()['id'];
    ...
}

```

```

/**
 * create one Layer (201)
 * @group testPost
 */
public function testCreateLayer(): void
{
    //-----
    // initialize required fields with the random values depending on their types
    //-----
    $LayerName = $this->generateRandomString();

    //-----
    // initialize required fields which are references to other entities
    //-----
    $response = $this->createClientWithCredentials('/api/metadata_sheets');
    $this->assertResponseIsSuccessful();
    $requiredRefItem = '/api/metadata_sheets' . '/' .
        strval($response->toArray()['hydra:member'][0]['id']);
    $LayerSheetMetadata = $requiredRefItem;

    $response = $this->createClientWithCredentials('/api/lex_layer_types');
    $this->assertResponseIsSuccessful();
    $requiredRefItem = '/api/lex_layer_types' . '/' .
        strval($response->toArray()['hydra:member'][0]['id']);
    $LayerLexLayerType = $requiredRefItem;

    //-----
    // create Layer with only required fields
    //-----
    $headers = ['content-type' => 'application/ld+json'];
    $body = [
        'name' => $LayerName,
        'sheetMetadata' => $LayerSheetMetadata,
        'lexLayerType' => $LayerLexLayerType,
    ];

    $response = $this->createClientWithCredentials($this->url, 'POST', $headers,
    $body);
    $this->assertResponseStatusCodeSame(201);

    self::$ItemId = $response->toArray()['id'];
    ...
}

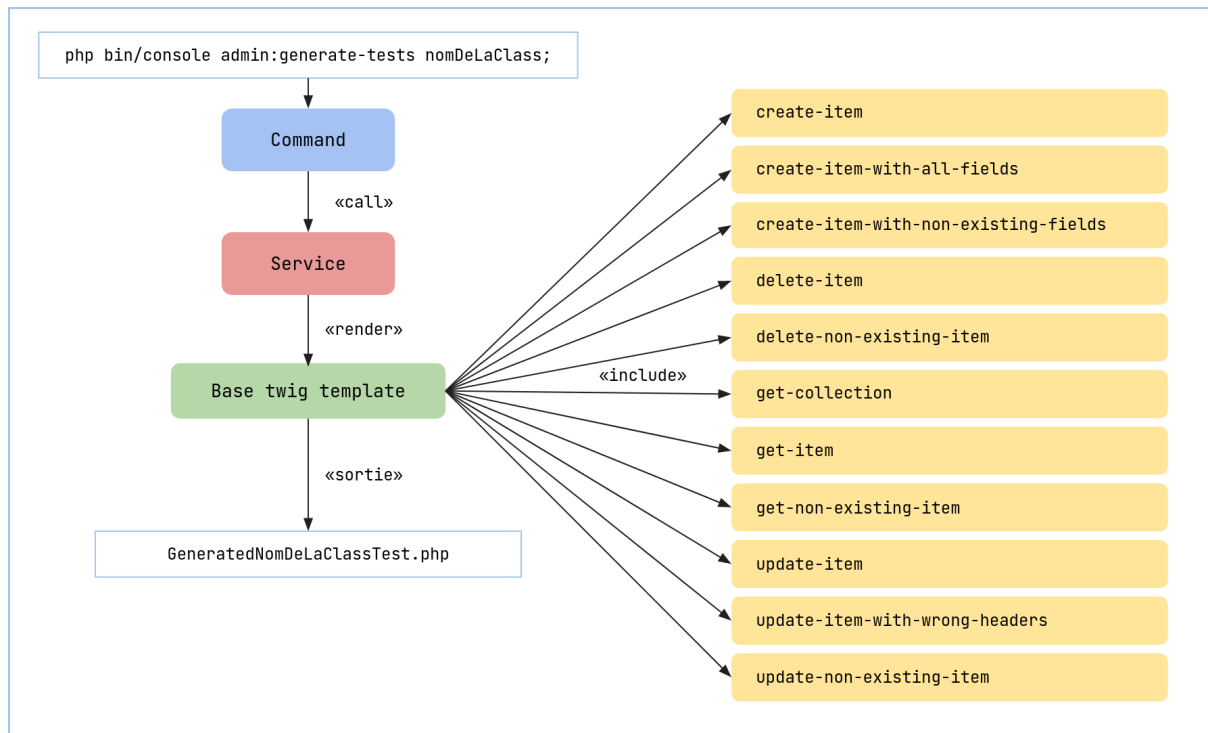
```

Le fait que la structure des tests ne soit pas significativement différente m'a conduit à l'idée que les tests pouvaient être paramétrés. Je suis également arrivé à la conclusion qu'écrire les mêmes tests pour différentes classes à partir de zéro est une action inutile. Copier et coller les mêmes morceaux de code pourrait conduire à des erreurs imprévues. En conséquence, j'ai suggéré à mon mentor de changer le sujet du stage et d'écrire une fonctionnalité pour générer automatiquement des tests basés sur les paramètres des classes.

3.4. Structure du générateur de test

Le projet de générateur de tests se compose de plusieurs éléments : la commande, le service, le modèle Twig de base et les modèles de tests.

La structure du générateur de test est la suivante :



L'ordre dans lequel le générateur fonctionne est le suivant :

1. Le développeur entre une commande pour générer des tests avec le nom de la classe
2. La commande class génère un schéma Json pour la classe afin d'en extraire les paramètres. En fonction des options de la commande saisie, la génération du schéma se fait différemment :

a.

```
php bin/console admin:generate-tests nomDeLaClass
```

Le schéma Json est généré à l'aide de commandes pour chaque type de schéma séparément. Par exemple, la commande suivante est utilisée pour la sortie du schéma GET :

```
$getOutputCollection = new ArrayInput([
    'command' => 'api:json-schema:generate',
    'resource' => $pathToResource,
    '--collectionOperation' => 'get',
    '--type' => 'output',
]);
```

b.

```
php bin/console admin:generate-tests nomDeLaClass -exp
```


Le diagramme Json est généré à l'aide d'une commande :

```
$schemaCommand = new ArrayInput([  
    'command' => 'api:openapi:export',  
]);
```

3. Le schéma Json est envoyé au service
4. Le service récupère les paramètres nécessaires du schéma, tels que les noms des champs, les types de données, la présence et le nombre de champs obligatoires, etc
5. Le service transmet les paramètres au modèle Twig de base, qui comprend également des modèles de test
6. Les modèles intègrent les paramètres dans le code des futurs tests
7. Le service rend les modèles et enregistre le code fini dans le dossier souhaité
8. Lorsque les tests sont générés, le développeur peut les exécuter à l'aide de la commande

```
php bin/phpunit pathToTestClass;
```

3.5. Options de commande

La commande de génération de tests contient plusieurs options, présentées ci-dessous :

```
Options:  
-u, --url[=URL]           API url  
-c, --defaultPathToDocs   Path to documentation file from config  
-p, --pathToDocs[=PATHTODOCS] Path to documentation file  
-d, --defaultFolder       Path to folder with classes to test from config  
-f, --folder[=FOLDER]     Path to folder with classes to test  
-t, --filter[=FILTER]     Filter classes to test
```

3.6. Création d'un bundle

Il a été décidé de créer un bundle afin que le projet soit disponible pour les développeurs de l'entreprise dans tous les projets. Les composants du projet ont été déplacés dans un projet séparé et

téléchargés sur GitLab. La classe TestGeneratorBundle.php et le fichier composer.json ont également été créés. Un lien a été établi entre le projet sur GitLab et le dépôt Composer d'Alkante. Ainsi, le package du projet peut maintenant être installé en utilisant la commande :

```
composer require prodige/test-generator:dev-develop;
```

Après avoir saisi la commande, le package du générateur de test est installé dans le dossier du vendeur. La commande de génération de tests est alors disponible dans le projet.

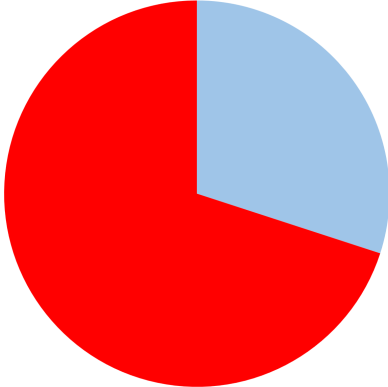
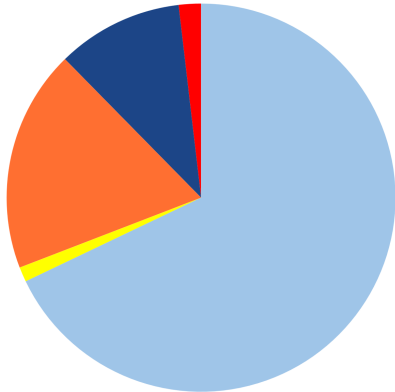
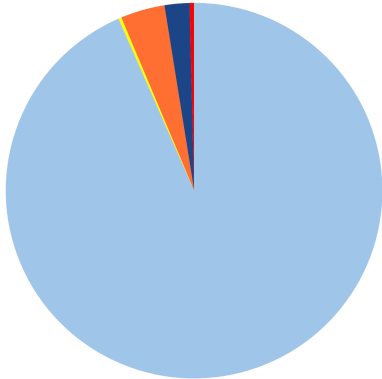
3.7. Mise en œuvre du projet

Pour mettre en œuvre le projet, le code a été envoyé sur GitLab. En outre, le projet a été mis en œuvre dans le système Jenkins, l'ordonnanceur de tâche de l'entreprise. À ce stade, avant qu'un développeur puisse ajouter une nouvelle version du code à la branche principale, Jenkins exécute des tests pour la classe modifiée. Si l'un des tests échoue, le système n'acceptera pas les modifications apportées par le développeur.

3.8. Résultats du générateur de test

J'ai testé 20 classes. Le nombre de tests est de 162 et le nombre d'assertions est de 791.

Suite au travail que j'ai effectué, j'ai compilé un rapport de bogues, que vous pouvez consulter dans annexe numéro 2. Quelques statistiques sont présentées ci-dessous :

<p>Classes :</p> <ul style="list-style-type: none"> - Ok : 6 - Error : 14 	<p> ● OK ● ERROR </p> 
<p>Tests</p> <ul style="list-style-type: none"> - Ok : 111 - Warnings : 2 - Skipped : 30 - Failures : 15 - Errors : 4 	<p> ● OK ● WARNINGS ● SKIPPED ● FAILURES ● ERRORS </p> 
<p>Assertions</p> <ul style="list-style-type: none"> - Ok : 740 - Warnings : 2 - Skipped : 30 - Failures : 15 - Errors : 4 	<p> ● OK ● WARNINGS ● SKIPPED ● FAILURES ● ERRORS </p> 

Les erreurs typiques comprennent les annotations `#[Assert\NotNull]` ou `#[Assert\NotBlank]` manquantes, la non-concordance entre le schéma d'entrée et le schéma de sortie, l'absence d'initialisation des champs.

4. Poursuite du développement du projet de stage

Bien que le projet de générateur de tests soit mis en œuvre et fonctionne, je vois plusieurs améliorations possibles. Tout d'abord, en plus des 11 modèles de test existants, de nouveaux modèles peuvent être ajoutés au projet. Grâce à la flexibilité du projet, le développeur n'a qu'à créer un modèle et à ajouter son nom au modèle de base. Deuxièmement, le développeur peut étendre le fichier de configuration afin de paramétrer davantage le projet.

5. Bilan du stage

Le stage chez Alkante m'a fait une impression très positive. J'ai été particulièrement impressionnée par la flexibilité de mes actions. Mon avis a été pris en compte concernant les idées pour le projet, et même l'objectif du stage a été modifié. Je ne peux pas oublier de mentionner également l'attention et le professionnalisme de mes mentors Anthony SUZANNE et Olivier BEDEL. Leur méthodologie de formation des stagiaires m'a été proche, et la communication a toujours été amicale.

En outre, j'ai apprécié les tâches qui m'ont été confiées au cours de mon stage. De plus, elles étaient importantes pour le développement de l'entreprise.

Les compétences que j'ai développées au cours de mon stage comprennent la créativité, la responsabilité et les compétences en communication. J'ai également acquis de nouvelles compétences dans le domaine du développement Symfony, des tests PHPUnit, de l'intégration continue, etc. Enfin, j'ai pratiqué le français en communiquant avec mes collègues.

RÉSUMÉ

Dans le cadre de mon stage au sein du Pôle géomatique d'Alkante, ma tâche principale a été d'écrire des tests pour les classes de la plateforme Prodiges. Durant les deux premières semaines, j'ai écrit 11 types de tests pour plusieurs classes. Au cours du processus de développement, j'ai remarqué des similitudes structurelles dans les tests des différentes classes, ce qui a été la base pour proposer le développement d'un générateur de tests qui automatiserait le processus d'écriture des tests.

Le générateur de tests se compose de plusieurs éléments : une commande, un service, un modèle Twig de base et des modèles de tests. Il permet de générer des diagrammes de classe JSON, d'en extraire des paramètres et de les insérer dans des modèles de test. La sortie du générateur est le code de test qui peut être exécuté avec PHPUnit.

Pendant le stage, le générateur a permis de générer 791 assertions dans le cadre de 162 tests effectués dans 20 classes.

Mon travail a été couronné de succès et le générateur de tests a été implémenté dans le projet. Il a considérablement simplifié le processus d'écriture et de maintenance des tests pour les classes d'API, ce qui a permis aux développeurs de gagner du temps et de réduire le risque d'erreurs.

Il a été constaté que 70 % des classes testées contenaient des bogues et un rapport de bogue a été généré.

Je considère mon expérience de stage chez Alkante comme positive. J'ai développé des compétences telles que la créativité, la responsabilité et la communication. J'ai également pu élargir considérablement mes compétences professionnelles en matière de développement Symfony, de tests PHPUnit et d'intégration continue.

ANNEXES

Annexe 1. Schéma d'entrée et de sortie de la classe Json

Schéma d'entrée d'une requête POST	Schéma de sortie d'une requête POST
<pre> ^ array:3 ["\$schema" => "http://json-schema.org/draft-07/schema#" "\$ref" => "#/definitions/Domain.jsonld-InputDomain" "definitions" => array:1 ["Domain.jsonld-InputDomain" => array:4 ["type" => "object" "description" => "Admin.Domain" "required" => array:2 [0 => "name" 1 => "rubric"] "properties" => array:7 ["@context" => array:2 ["readOnly" => true "oneOf" => array:2 [0 => array:1 ["type" => "string"] 1 => array:4 ["type" => "object" "properties" => array:2 ["@vocab" => array:1 ["type" => "string"]] "hydra" => array:2 ["type" => "string" "enum" => array:1 [0 => "http://www.w3.org/ns/hydra/core#"]]]]] "required" => array:2 [0 => "@vocab" 1 => "hydra"] "additionalProperties" => true]]] </pre>	<pre> ^ array:3 ["\$schema" => "http://json-schema.org/draft-07/schema#" "\$ref" => "#/definitions/Domain.jsonld-GetDomain" "definitions" => array:2 ["Domain.jsonld-GetDomain" => array:4 ["type" => "object" "description" => "Admin.Domain" "properties" => array:10 ["@context" => array:2 ["readOnly" => true "oneOf" => array:2 [0 => array:1 ["type" => "string"] 1 => array:4 ["type" => "object" "properties" => array:2 ["@vocab" => array:1 ["type" => "string"]] "hydra" => array:2 ["type" => "string" "enum" => array:1 [0 => "http://www.w3.org/ns/hydra/core#"]]]]] "required" => array:2 [0 => "@vocab" 1 => "hydra"] "additionalProperties" => true]]] "id" => array:2 ["readOnly" => true "type" => "string"] "@type" => array:2 [</pre>

<pre> "@id" ⇒ array:2 ["readOnly" ⇒ true "type" ⇒ "string"] "@type" ⇒ array:2 ["readOnly" ⇒ true "type" ⇒ "string"] "name" ⇒ array:1 ["type" ⇒ "string"] "description" ⇒ array:1 ["type" ⇒ array:2 [0 ⇒ "string" 1 ⇒ "null"]] "rubric" ⇒ array:2 ["type" ⇒ "string" "format" ⇒ "iri-reference"] "subdomains" ⇒ array:3 ["readOnly" ⇒ true "type" ⇒ "array" "items" ⇒ array:2 ["type" ⇒ "string" "format" ⇒ "iri-reference"]]]]]] </pre>	<pre> "readOnly" ⇒ true "type" ⇒ "string"] "id" ⇒ array:2 ["readOnly" ⇒ true "type" ⇒ "integer"] "name" ⇒ array:1 ["type" ⇒ "string"] "description" ⇒ array:1 ["type" ⇒ array:2 [0 ⇒ "string" 1 ⇒ "null"]] "rubric" ⇒ array:1 ["\$ref" ⇒ "#/definitions/Rubric.jsonld-GetDomain"] "subdomains" ⇒ array:3 ["readOnly" ⇒ true "type" ⇒ "array" "items" ⇒ array:2 ["type" ⇒ "string" "format" ⇒ "iri-reference"]]] "countSubdomain" ⇒ array:2 ["readOnly" ⇒ true "type" ⇒ array:2 [0 ⇒ "integer" 1 ⇒ "null"]]] "canDelete" ⇒ array:2 ["readOnly" ⇒ true "type" ⇒ "boolean"]] "required" ⇒ array:2 [0 ⇒ "name" 1 ⇒ "rubric"]] "Rubric.jsonld-GetDomain" ⇒ array:4 ["type" ⇒ "object" "description" ⇒ "Admin.Rubric" "required" ⇒ array:1 [0 ⇒ "name"] "properties" ⇒ array:4 ["@context" ⇒ array:2 ["readOnly" ⇒ true "oneOf" ⇒ array:2 [</pre>
---	--

```
0 => array:1 [
    "type" => "string"
]
1 => array:4 [
    "type" => "object"
    "properties" => array:2 [
        "@vocab" => array:1 [
            "type" => "string"
        ]
        "hydra" => array:2 [
            "type" => "string"
            "enum" => array:1 [
                0 =>
"http://www.w3.org/ns/hydra/core#"
            ]
        ]
    ]
    "required" => array:2 [
        0 => "@vocab"
        1 => "hydra"
    ]
    "additionalProperties" =>
true
]
]
"@id" => array:2 [
    "readOnly" => true
    "type" => "string"
]
"@type" => array:2 [
    "readOnly" => true
    "type" => "string"
]
"name" => array:1 [
    "type" => "string"
]
]
```


Annexe 2. Rapport de bogue

Nº	Lieu	Chemin	Description	Correction
1	Layer	App\Entity\Administration\Resource\Layer;	Property 'name' is not unique. So that the same name can be used many times.	add #[UniqueEntity(fields: 'name')]
2	Layer	App\Entity\Administration\Resource\Layer;	Property 'name' can be blank, but has 'nullable: false' at the same time. It leads to contradiction. In this case they are not required in JSON schema, but still must be defined.	add #[Assert\NotBlank] for 'name' in Layer.php (line 88)
3	Layer	App\Entity\Administration\Resource\Layer;	Properties 'sheetMetadata' and 'lexLayerType' are in patch output group, but aren't in patch input group.	add #[Groups(['PatchLayer'])]; for 'sheetMetadata' and 'lexLayerType'
4	Domain	App\Entity\Administration\Organisation\Domain;	There's no setter for property 'subdomain'.	add public function setSubdomain(...) {...}
5	Map	App\Entity\Administration\Resource\Map;	Properties 'name' and 'path' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Validator\Constraints as Assert; add #[Assert\NotNull] for 'name' and 'path'
6	MetadataSheet	App\Entity\Administration\Resource\MetadataSheet;	Property 'name' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Validator\Constraints as Assert; add #[Assert\NotNull] for 'name'
7	Format	App\Entity\Configuration\Format;	Property 'name' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Validator\Constraints as Assert; add #[Assert\NotNull] for 'name'
8	Module	App\Entity\Configuration\Module;	Properties 'name' and 'path' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Validator\Constraints as Assert; add #[Assert\NotNull] for 'name' and 'path'

9	Projection	App\Entity\Configurati ion\Projection;	Properties 'name' and 'epsg' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Valid ator\Constraints as Assert; add #[Assert\NotNull] for 'name' and 'epsg'
10	Skill	App\Entity\Administra tion\Directory\Skill;	Property 'name' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Valid ator\Constraints as Assert; add #[Assert\NotNull] for 'name'
11	Rubric	App\Entity\Administra tion\Organisation\Rub ric;	Property 'name' is not unique. So that the same name can be used many times.	add #[UniqueEntity(fields: 'name')]
12	Setting	App\Entity\Configurati on\Setting;	Une valeur NULL viole la contrainte NOT NULL de la colonne «description».	change 'nullable: false' to 'nullable: true' for property 'description'
13	Setting	App\Entity\Configurati on\Setting;	\$projection, \$formats and \$templates are not readable.	add public function __construct() { \$this->projections = new ArrayCollection(); \$this->formats = new ArrayCollection(); \$this->templates = new ArrayCollection(); }
14	Setting	App\Entity\Configurati on\Setting;	There's no type definitions for properties.	add #[Assert\Type('string')]
15	Setting	App\Entity\Configurati on\Setting;	Properties 'name' and 'title' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Valid ator\Constraints as Assert; add #[Assert\NotNull] for 'name' and 'title'
16	Structure	App\Entity\Administra tion\Directory\Struct ure;	Property 'name' can be null, but has 'nullable: false' at the same time. It leads to contradiction.	use Symfony\Component\Valid ator\Constraints as Assert; add #[Assert\NotNull] for 'name'
17	Structure	App\Entity\Administra tion\Directory\Struct ure;	There's no denormalization context.	add normalizationContext: ['groups' => [...]]
18	Subdomain	App\Entity\Administra tion\Organisation\Sub domain;	profileCreator: NULL value found, but an object is required. profileCreator: Failed to match at least one	?

			schema.	
19	User	App\Entity\Administration\Directory\User\User;	Syntax error in GeonetworkService.php.	?
20	User	App\Entity\Administration\Directory\User\User;	Admin cannot be null.	private User \$admin ⇒ private ?User \$admin in User.php (line 146)
21	Zonning	App\Entity\Administration\Directory\Zoning;	\$areas is not readable.	add public function __construct() { \$this->areas = new ArrayCollection(); }
22	CurlUtilities	site/vendor/prodige/core/Services/CurlUtilities.php	Problem with CAS.	add CurlUtilities.php if (empty(\$basicAuthentication)) { \$basicAuthentication = array(\$_ENV['PHPCLI_DEFAULT_LOGIN'], \$_ENV['PHPCLI_DEFAULT_PASSWORD']); }